1,项目开发流程

- 目的: 了解项目的开发流程
- 开发流程:
    - 立项 -> 需求分析和设计(原型图) -> 开发(后端,前端,美工) -> 代码整合(测试) ->上线

2,项目架构设计

- 目的: 常见的两种开发方式, 以及美多商城前台项目的技术选型
- 常见的两种开发方式:
    - 1, 前后端不分离 (美多商城前台)
    - 2, 前后端分离
- 美多商城前台技术选型:
    - 1, 开发模式
    - 2, 技术: Django + jinja2
    - 3, 前端: Vue.js(管理页面 + 网络请求axios.js)

3,项目创建(码云)

- 目的: 能够使用码云管理美多商城项目
- 操作流程:
    - 1, 在码云创建一个仓库
    - 2, clone到本地
    - 3, 在仓库中创建美多项目
    - 4, 推送到码云中即可

4,开发环境配置(settings.py)

- 目的: 能够配置两套运行配置环境
- 操作流程:
    - 1, 在根应用的下面创建了settings文件夹
    - 2, 在里面, 创建了两份环境(dev.py ,prod.py)
    - 3, 修改manage.py的启动配置项

```
os.environ.setdefault("DJANGO_SETTINGS_MODULE", "meiduo_mall.settings.dev")
```

5,jinja2模板引擎

- 目的: 能够配置django中的模板引擎为Jinja2
- 使用流程:
  - 1, 现在根路径下面创建templates文件夹
  - 2, 在dev.py中指定Jinja2的模板

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.jinja2.Jinja2', #1
        'DIRS': [os.path.join(BASE_DIR,'templates')], #2
        'APP_DIRS': True,
        'OPTIONS': {
            'environment':'meiduo_mall.utils.jinja2_env.environment', #3指定模块加载的环境
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

- 3, 在根路径下面创建utils/jinja2.py, 编写加载模板的方法(来自于官方文档)

```
from django.contrib.staticfiles.storage import staticfiles_storage
from django.urls import reverse

from jinja2 import Environment


def environment(**options):
    env = Environment(**options)
    env.globals.update({
        'static': staticfiles_storage.url,
        'url': reverse,
    })
    return env
```

6,mysql配置

- 目的: 能够根据文档配置mysql的存储
- 配置流程:
  - 1, 设置数据库的链接信息(来自于文档)

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'meiduo12',
        'USER': 'root',
        'PASSWORD': '123456',
        'HOST': '127.0.0.1',
        'PORT': '3306',
    }
}
```

  - 2, 指定mysql的驱动(meiduo_mall/init.py)

```
import pymysql
pymysql.install_as_MySQLdb()
```

- 3, 创建数据库

7,配置redis数据库

- 目的: 能够参考django-redis文档,配置redis
  - 配置(参考了django-redis)

```
CACHES = {
    "default": {
        "BACKEND": "django_redis.cache.RedisCache",
        "LOCATION": "redis://127.0.0.1:6379/0",
        "OPTIONS": {
            "CLIENT_CLASS": "django_redis.client.DefaultClient",
        }
    },
    "session": {
        "BACKEND": "django_redis.cache.RedisCache",
        "LOCATION": "redis://127.0.0.1:6379/1",
        "OPTIONS": {
            "CLIENT_CLASS": "django_redis.client.DefaultClient",
        }
    }
}

SESSION_ENGINE = "django.contrib.sessions.backends.cache"
SESSION_CACHE_ALIAS = "default"
```

8,日志工程的配置

- 目的: 能够参考文档配置日志
- 配置(django文档):

```
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,  # 是否禁用已经存在的日志器
    'formatters': {  # 日志信息显示的格式
        'verbose': {
            'format': '%(levelname)s %(asctime)s %(module)s %(lineno)d %(message)s'
        },
        'simple': {
            'format': '%(levelname)s %(module)s %(lineno)d %(message)s'
        },
    },
    'filters': {  # 对日志进行过滤
        'require_debug_true': {  # django在debug模式下才输出日志
            '()': 'django.utils.log.RequireDebugTrue',
        },
    },
    'handlers': {  # 日志处理方法
        'console': {  # 向终端中输出日志
            'level': 'INFO',
            'filters': ['require_debug_true'],
            'class': 'logging.StreamHandler',
            'formatter': 'simple'
        },
        'file': {  # 向文件中输出日志
            'level': 'INFO',
            'class': 'logging.handlers.RotatingFileHandler',
            'filename': os.path.join(os.path.dirname(BASE_DIR), 'logs/meiduo.log'),  # 日志文件的位置
            'maxBytes': 100 * 1024 * 1024,
            'backupCount': 10,
            'formatter': 'verbose'
        },
```

```
    },
    'loggers': {  # 日志器
        'django': {  # 定义了一个名为django的日志器
            'handlers': ['console', 'file'],  # 可以同时向终端与文件中输出日志
            'propagate': True,  # 是否继续传递日志信息
            'level': 'INFO',  # 日志器接收的最低日志级别,INFO < debug < warn < error
        },
    }
}
```

9,静态文件集成

- 目的: 能够将静态文件集成到项目中
- 过程:
  - 在项目的根目录下面, 添加static静态文件夹

10,创建用户模块子应用

- 目的: 能够通过命令创建用户子应用在对应的文件夹下
- 过程:
  - 1, 在根应用下面创建apps文件夹
  - 2, 进入apps中,创建子应用
    - python ../../manage.py startapp users
- 目的:
  - 方便以后扩展使用

11,设置导包路径

- 目的: 能够设置apps作为新的导包路径
- 操作流程:

```
import sys
#告诉系统apps作为了子应用的新的导包路径
sys.path.insert(0,os.path.join(BASE_DIR,'apps'))
```

```
# print(sys.path)

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'users.apps.UsersConfig',
]
```

12,展示用户注册页面

- 目的: 能够定义视图显示注册页面
- 操作流程:
  - 1, 将register.html移动到templates中
  - 2, 编写类视图

```
class RegisterView(View):
    def get(self,request):
        return render(request,'register.html')

```

  - 3, 编写子应用的urls.py

```
from django.conf.urls import url
from . import views

urlpatterns = [

 url(r'^register/$',views.RegisterView.as_view(),name="register")
]

```

  - 4, 编写根应用urls.py的路径

```
urlpatterns = [
    ...
    url(r'^users/',
include('users.urls',namespace="user"))
]

```

13,用户模型类创建

- 目的: 能够定义模型类,继承自系统的AbstractUser类
- 定义格式:
    - 1, 在users/models中定义User继承AbstractUser

```
from django.db import models
from django.contrib.auth.models import
AbstractUser

class User(AbstractUser):
    mobile =
models.CharField(max_length=11,unique=True,ver
bose_name="手机号")

    class Meta:
        db_table = "tb_users"

    def __str__(self):
        return self.username


```

    - 2,在dev.py中设置,用户模型类

```
#指定用户模型类
AUTH_USER_MODEL = 'users.User'
```

- 好处:
    - 1,继承自系统的Abstrack之后, 已经提供了通用的属性
    - 2, 提供了常用的方法,比如密码加密, 密码的校验

14,用户模型类迁移

- 目的: 能够迁移模型类的内容,到数据库中
- 命令格式:
    - python manage.py makemigrations
    - python manage.py migrate

15,代码块配置

- 目的: 将一些常见的代码,配置成模版, 方便我们编写程序